Tutorial is limited to pyspark transition to pandas users. Databricks would be seperate

In [50]:

```
import pandas as pd
from pyspark.sql import SparkSession
from pyspark import SparkContext
spark = SparkSession.builder.appName('SparkApp').getOrCreate()
```

Pyspark includes

- Native query

Pyspark doesn't include

- Indices

Hadoop Consistes of 3 layers

- Hdfs : file storage
- Spark : for processing and manipulation of data (replacement of mapreduce)
- Yarn : resource management

In [51]:

```
from pyspark.sql import functions as f
from pyspark.sql.functions import col, concat, lit
from pyspark.sql.types import IntegerType,BooleanType,DateType,StringType,DoubleType
import numpy as np
```

In [52]:

```
# read file in pyspark

df_spark = spark.read.options(header='True', inferSchema='True', delimiter=',') \
  .csv("test1.csv")

#header = true - assumes header - first row
#inferSchema - detects data type
#delimiter - specifying delimiter

df_spark.show(1)
```

```
+-----+---+----------+------+
| Name|age|Experience|Salary|
+-----+---+----------+------+
|Krish| 31|        10| 30000|
+-----+---+----------+------+
only showing top 1 row
```

In [53]:

```
# read file in pandas

df_pandas = pd.read_csv('test1.csv', sep = ',')
df_pandas.head(1)
```

Out[53]:

	Name	age	Experience	Salary
0	Krish	31	10	30000

In [54]:

```
# writing in pyspark

#df_spark.write.options(header='True', delimiter=',').csv("output")
#df_spark.write.format("csv").save("output")
#df_spark.write.csv("test.csv")
#.mode('overwrite')
#df_spark.coalesce(1).write.csv('result.csv')
#df_spark.coalesce(1).write.csv("header.csv", header="true")


#df.write() API will create multiple part files inside given path
#to force spark write only a single part file use df.coalesce(1).write.csv(...)
#instead of df.repartition(1).write.csv(...) as coalesce is a narrow transformation whe
reas repartition is a wide transformation
```

In [56]:

```
# writing in pandas

#df.to_csv('file.csv', index=None)
```

In [57]:

```
## Convert to pandas from spark
pd_df = df_spark.toPandas()
## Convert into Spark DataFrame
spark_df = spark.createDataFrame(pd_df)
```

In [ ]:

```
#spark save as delta table
spark_df.write.mode("overwrite").saveAsTable("temp.eehara_trial_table_9_5_19")

#you can create a new pandas dataframe witht the following command:
pd_df = spark.sql('select * from temp.eehara_trial_table_9_5_19').toPandas()

#spark query the delta table----------------------------------------

%sql

select * from temp.eehara_trial_table_9_5_19 ;
```

In [61]:

```
# view dataframe

#pandas

print(df_pandas.head(1))  # first 10
df_pandas.tail(5) #last 10

#spark

df_spark.show(1) #shows 1 rows
df_spark.limit(1) #limit 1 first rows
#df_spark.display() # to download - specific to databrics
```

```
    Name  age  Experience  Salary
0  Krish   31          10   30000
+-----+---+----------+------+
| Name|age|Experience|Salary|
+-----+---+----------+------+
|Krish| 31|        10| 30000|
+-----+---+----------+------+
only showing top 1 row
```

Out[61]:

```
DataFrame[Name: string, age: int, Experience: int, Salary: int]
```

In [62]:

```
# pandas columns and data types

df_pandas.columns
df_pandas.dtypes

# pyspark columns and data types

df_spark.columns
df_spark.dtypes
```

Out[62]:

```
[('Name', 'string'), ('age', 'int'), ('Experience', 'int'), ('Salary', 'in
t')]
```

Renaming columns

In [63]:

```
#Pandas

df_pandas.rename(columns = {"Experience":"Exp"})

#pyspark

df_spark.withColumnRenamed("Experience","Exp")
```

Out[63]:

```
DataFrame[Name: string, age: int, Exp: int, Salary: int]
```

Dropping columns

In [64]:

```
#pandas

df_pandas.drop('Experience', axis = 1)

#spark

df_spark.drop('Experience')
```

Out[64]:

```
DataFrame[Name: string, age: int, Salary: int]
```

Filtering

In [65]:

```
# pandas

df_pandas[df_pandas['age'] > 10]
df_pandas[(df_pandas['age'] > 20) & (df_pandas['Experience'] > 5)]

#spark

df_spark[df_spark['age'] > 10]
df_spark[(df_spark['age'] > 20) & (df_spark['Experience'] > 5)]
```

Out[65]:

```
DataFrame[Name: string, age: int, Experience: int, Salary: int]
```

In [66]:

```
#in pyspark you can also do

### Salary of the people less than or equal to 20000
#df_spark.filter("Experience<=20").show()

#and

df_spark.filter(~(df_spark['Salary']<=27000) |
                  (df_spark['age']>= 31)).show()
```

```
+-----+---+----------+------+
| Name|age|Experience|Salary|
+-----+---+----------+------+
|Krish| 31|        10| 30000|
+-----+---+----------+------+
```

adding column

In [67]:

```
# pandas

df_pandas['retirement age'] = df_pandas['age'] + 3

#pyspark

df_spark.withColumn('retirement age', df_spark.age + 3)
```

Out[67]:

```
DataFrame[Name: string, age: int, Experience: int, Salary: int, retirement
age: int]
```

In [68]:

```
# filling nulls # can be used for whole df

# pandas

df_pandas['age'].fillna(0)

#spark

df_spark['age'].fillna(0)  #oops
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-68-5e9b5739d410> in <module>
      7 #spark
      8
----> 9 df_spark['age'].fillna(0)  #oops

TypeError: 'Column' object is not callable
```

In [72]:

```
# using select function from pyspark

df_spark.select(f.col('age')).fillna(0)
#or
df_spark.withColumn('age',f.col('age')).fillna(0).show(1)
```

```
+-----+---+----------+------+
| Name|age|Experience|Salary|
+-----+---+----------+------+
|Krish| 31|        10| 30000|
+-----+---+----------+------+
only showing top 1 row
```

group bys

In [76]:

```
df_pandas.groupby(['Name']).agg({'age':'max'})[0:1]

#or

#df_pandas.groupby(['Name'], as_index = False)['age'].max()

#which is same as

#df_pandas.groupby(['Name'])['age'].max().reset_index()
```

Out[76]:

	age
Name	
Harsha	21

In [78]:

```
#spark

df_spark.groupby(['Name']).agg({'age':'max'}).show(1)
```

```
+---------+--------+
|     Name|max(age)|
+---------+--------+
|Sudhanshu|      30|
+---------+--------+
only showing top 1 row
```

renaming columns after groupby

In [81]:

```
#pandas

df_pandas.groupby(['Name'], as_index=False).agg({'age':'max'}).rename(columns = {"Exper
ience":"Exp"}).head(1)

#to keep all columns you must specify aggregation for those columns too or group by the
m
```

Out[81]:

	Name	age
0	Harsha	21

In [82]:

```
#pyspark

df_spark.groupby(['Name']).agg({'age':'max'}).withColumnRenamed("max(age)","age").show(
1)
```

```
+---------+---+
|     Name|age|
+---------+---+
|Sudhanshu| 30|
+---------+---+
only showing top 1 row
```

conditionals statements

In [83]:

```
#pandas

df_pandas['eligible'] = np.where(df_pandas['age'] > 25, "Yes", "No")
df_pandas.head(2)
```

Out[83]:

	Name	age	Experience	Salary	retirement age	eligible
0	Krish	31	10	30000	34	Yes
1	Sudhanshu	30	8	25000	33	Yes

In [84]:

```
#pyspark

df_spark.withColumn('eligible', f.when(df_spark['age'] > 30, "yes").when(df_spark['Sala
ry'] > 0, "Yip").otherwise("No")).show(2)
```

```
+---------+---+----------+------+--------+
|     Name|age|Experience|Salary|eligible|
+---------+---+----------+------+--------+
|    Krish| 31|        10| 30000|     yes|
|Sudhanshu| 30|         8| 25000|     Yip|
+---------+---+----------+------+--------+
only showing top 2 rows
```

In [85]:

```
#there are bunch of ways in pandas for multiple conditions, below in one another exampl
e

##df = pd.DataFrame({'Type':list('ABBC'), 'Set':list('ZZXY')})
##conditions = [
##    (df['Set'] == 'Z') & (df['Type'] == 'A'),
##    (df['Set'] == 'Z') & (df['Type'] == 'B'),
##    (df['Type'] == 'B')]
##choices = ['yellow', 'blue', 'purple']
##df['color'] = np.select(conditions, choices, default='black')
##print(df)
```

lambda functions

In [86]:

```
#pandas

df_pandas['extra bonus'] = df_pandas['Salary'].apply(lambda x : x+ 3000)
df_pandas.head(2)
```

Out[86]:

	Name	age	Experience	Salary	retirement age	eligible	extra bonus
0	Krish	31	10	30000	34	Yes	33000
1	Sudhanshu	30	8	25000	33	Yes	28000

In [87]:

```
#pyspark

fnctnwa = f.udf(lambda x : x + 3000)
df_spark.withColumn('extra bonus',  fnctnwa(df_spark.Salary)).show(2)
```

```
+---------+---+----------+------+-----------+
|     Name|age|Experience|Salary|extra bonus|
+---------+---+----------+------+-----------+
|    Krish| 31|        10| 30000|      33000|
|Sudhanshu| 30|         8| 25000|      28000|
+---------+---+----------+------+-----------+
only showing top 2 rows
```

extra

apply, applymap and map

- apply() is used to apply a function along an axis of the DataFrame or on values of Series.
- applymap() is used to apply a function to a DataFrame elementwise.
- map() is used to substitute each value in a Series with another value.
- df['D'] = df['D'].apply(lambda x:x.sum(), axis=1)
- df = df.applymap(lambda x:x.sum(), axis=1)
- df['D'] = df['C'].map(dictionary)

JOINS

pyspark

In [88]:

```
df1=spark.read.csv('test1.csv',header=True,inferSchema=True).limit(2)
df1.show()
df2=spark.read.csv('test1.csv',header=True,inferSchema=True).limit(2)
df2.show()
```

```
+---------+---+----------+------+
|     Name|age|Experience|Salary|
+---------+---+----------+------+
|    Krish| 31|        10| 30000|
|Sudhanshu| 30|         8| 25000|
+---------+---+----------+------+

+---------+---+----------+------+
|     Name|age|Experience|Salary|
+---------+---+----------+------+
|    Krish| 31|        10| 30000|
|Sudhanshu| 30|         8| 25000|
+---------+---+----------+------+
```

In [89]:

```
##1. when column name are same and you dont want duplicate column names

df1.join(df2, on='Name').show()
```

```
+---------+---+----------+------+---+----------+------+
|     Name|age|Experience|Salary|age|Experience|Salary|
+---------+---+----------+------+---+----------+------+
|    Krish| 31|        10| 30000| 31|        10| 30000|
|Sudhanshu| 30|         8| 25000| 30|         8| 25000|
+---------+---+----------+------+---+----------+------+
```

In [90]:

```
# multiple columns , same name

df1.join(df2, on=['Name','age'], how='left').show()
```

```
+---------+---+----------+------+----------+------+
|     Name|age|Experience|Salary|Experience|Salary|
+---------+---+----------+------+----------+------+
|    Krish| 31|        10| 30000|        10| 30000|
|Sudhanshu| 30|         8| 25000|         8| 25000|
+---------+---+----------+------+----------+------+
```

In [91]:

```
#left on, right on for different column names wont work.. so need to rename column with
above technique

df2 = df2.withColumnRenamed('Name','Name2').withColumnRenamed('Experience','Experience
2')
df1.join(df2, ((df1.Name == df2.Name2) & (df1.Experience == df2.Experience2)), how = 'l
eft' ).show()
```

```
+---------+---+----------+------+---------+---+-----------+------+
|     Name|age|Experience|Salary|    Name2|age|Experience2|Salary|
+---------+---+----------+------+---------+---+-----------+------+
|    Krish| 31|        10| 30000|    Krish| 31|         10| 30000|
|Sudhanshu| 30|         8| 25000|Sudhanshu| 30|          8| 25000|
+---------+---+----------+------+---------+---+-----------+------+
```

pandas

In [92]:

```
df1=pd.read_csv('test1.csv')
df2=pd.read_csv('test1.csv')
df2.head(2)
```

Out[92]:

	Name	age	Experience	Salary
0	Krish	31	10	30000
1	Sudhanshu	30	8	25000

In [93]:

```
df1.merge(df2, left_on=['Name'], right_on=['Name'], how = 'left').head(2)
```

Out[93]:

	Name	age_x	Experience_x	Salary_x	age_y	Experience_y	Salary_y
0	Krish	31	10	30000	31	10	30000
1	Sudhanshu	30	8	25000	30	8	25000

In [94]:

```
#or

df1.merge(df2, on=['Name'], how = 'left').head(2)
```

Out[94]:

	Name	age_x	Experience_x	Salary_x	age_y	Experience_y	Salary_y
0	Krish	31	10	30000	31	10	30000
1	Sudhanshu	30	8	25000	30	8	25000

pivot table

In [95]:

```
#pandas

pd.pivot_table(df1, values='Salary',index=
               'Name', columns='age', aggfunc= np.sum)
```

Out[95]:

age	21	23	24	29	30	31
Name						
Harsha	15000.0	NaN	NaN	NaN	NaN	NaN
Krish	NaN	NaN	NaN	NaN	NaN	30000.0
Paul	NaN	NaN	20000.0	NaN	NaN	NaN
Shubham	NaN	18000.0	NaN	NaN	NaN	NaN
Sudhanshu	NaN	NaN	NaN	NaN	25000.0	NaN
Sunny	NaN	NaN	NaN	20000.0	NaN	NaN

In [96]:

```
#pyspark

df_spark.groupby('Name').pivot('age').sum('Salary').show()
```

```
+---------+-----+-----+-----+-----+-----+-----+
|     Name|   21|   23|   24|   29|   30|   31|
+---------+-----+-----+-----+-----+-----+-----+
|Sudhanshu| null| null| null| null|25000| null|
|    Sunny| null| null| null|20000| null| null|
|    Krish| null| null| null| null| null|30000|
|   Harsha|15000| null| null| null| null| null|
|     Paul| null| null|20000| null| null| null|
|  Shubham| null|18000| null| null| null| null|
+---------+-----+-----+-----+-----+-----+-----+
```

ISIN

In [97]:

```
#pandas

df_pandas[df_pandas['Name'].isin(['Sunny'])]
```

Out[97]:

	Name	age	Experience	Salary	retirement age	eligible	extra bonus
2	Sunny	29	4	20000	32	Yes	23000

In [98]:

```
#spark #opposite # same

df_spark[~df_spark['Name'].isin(['Sunny'])].show(2)
```

```
+---------+---+----------+------+
|     Name|age|Experience|Salary|
+---------+---+----------+------+
|    Krish| 31|        10| 30000|
|Sudhanshu| 30|         8| 25000|
+---------+---+----------+------+
only showing top 2 rows
```

ISIN for vlookup -- passing columns

In [99]:

```
#pandas

df1=pd.read_csv('test1.csv')
df2=pd.read_csv('test1.csv')
df1[df1['Name'].isin(df2['Name'])].head(2)
```

Out[99]:

	Name	age	Experience	Salary
0	Krish	31	10	30000
1	Sudhanshu	30	8	25000

In [110]:

```
df1=spark.read.csv('test1.csv',header=True,inferSchema=True).limit(2)
df2=spark.read.csv('test1.csv',header=True,inferSchema=True).limit(2)
```

In [111]:

```
#df1[df1['Name'].isin(df2['Name'])] #error
```

In [112]:

```
#### we get error here

##soo

my_list = list(
    df2.select('Name').distinct().toPandas()['Name']
)


df1[df1['Name'].isin(my_list)].show()
```

```
+---------+---+----------+------+
|     Name|age|Experience|Salary|
+---------+---+----------+------+
|    Krish| 31|        10| 30000|
|Sudhanshu| 30|         8| 25000|
+---------+---+----------+------+
```

Union, concat, append

pandas specific

- Concat gives the flexibility to join based on the axis(all rows or all columns)
- Append is the specific case(axis=0, join='outer') of concat (being deprecated use concat)
- Join is based on the indexes (set by set_index) on how variable =['left','right','inner','couter']
- Merge is based on any particular column each of the two dataframes, like 'left_on', 'right_on', 'on'

In [132]:

```
#pyspark

df1=spark.read.csv('test1.csv',header=True,inferSchema=True).limit(2)
df2=spark.read.csv('test1.csv',header=True,inferSchema=True).limit(2)

#df1 =  df1.withColumn("Random", lit("haha"))  -- additional column fails
df1.limit(1).union(df2.limit(1)).show(2)

#pandas

df1=pd.read_csv('test1.csv')
df2=pd.read_csv('test1.csv')

print(pd.concat([df1,df2])[0:2])

print(df1.append(df2)[0:2])
```

```
+-----+---+----------+------+
| Name|age|Experience|Salary|
+-----+---+----------+------+
|Krish| 31|        10| 30000|
|Krish| 31|        10| 30000|
+-----+---+----------+------+

        Name  age  Experience  Salary
0      Krish   31          10   30000
1  Sudhanshu   30           8   25000
        Name  age  Experience  Salary
0      Krish   31          10   30000
1  Sudhanshu   30           8   25000
```

notes of union spark

- union() method merges two DataFrames and returns the new DataFrame with all rows
- unionAll() method is deprecated since PySpark “2.0.0”
- in sql unionall includes dupes, union doesn't
- Merge without Duplicates :
  - df.union(df2).distinct()

In [113]:

```
# Trim the spaces from both ends for the specified string column.

#pyspark

from pyspark.sql.functions import trim
df_spark = df_spark.withColumn("Name", trim(df_spark.Name))

#pandas

df_pandas = df_pandas.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

In [114]:

```
# upper or lower a dataframe

# pyspark
from pyspark.sql.functions import upper, lower

df_spark.withColumn("upper", upper(df_spark.Name)).withColumn(
    "lower", lower(df_spark.Name)
).show(1)

#pandas


df_pandas['Name'].str.upper()[0]
```

```
+-----+---+----------+------+-----+-----+
| Name|age|Experience|Salary|upper|lower|
+-----+---+----------+------+-----+-----+
|Krish| 31|        10| 30000|KRISH|krish|
+-----+---+----------+------+-----+-----+
only showing top 1 row
```

Out[114]:

```
'KRISH'
```

In [115]:

```
# adding default column

#pyspark

df_spark.withColumn("Sex",f.lit("Male")).show(1)

#pandas

df_pandas["Sex"] = "Male"

#or

from pyspark.sql.functions import lit

#df = auto_df.withColumn("one", lit(1)
```

```
+-----+---+----------+------+----+
| Name|age|Experience|Salary| Sex|
+-----+---+----------+------+----+
|Krish| 31|        10| 30000|Male|
+-----+---+----------+------+----+
only showing top 1 row
```

In [116]:

```
# adding 2 columns

#pyspark

from pyspark.sql.functions import col, concat, lit

df_spark.withColumn("Sex age", concat(lit("Male "), col("age"))).show(1)

#pandas

df_pandas['Sex age'] = "Male " + df_pandas['age'].astype('str')
df_pandas.head(1)
```

```
+-----+---+----------+------+-------+
| Name|age|Experience|Salary|Sex age|
+-----+---+----------+------+-------+
|Krish| 31|        10| 30000|Male 31|
+-----+---+----------+------+-------+
only showing top 1 row
```

Out[116]:

	Name	age	Experience	Salary	retirement age	eligible	extra bonus	Sex	Sex age
0	Krish	31	10	30000	34	Yes	33000	Male	Male 31

In [117]:

```
#Get the size of a DataFrame

#pyspark

print("{} rows".format(df_spark.count()))
print("{} columns".format(len(df_spark.columns)))

#pandas

df_pandas.shape
```

```
6 rows
4 columns
```

Out[117]:

```
(6, 9)
```

data type conversion

In [118]:

```
#pyspark

from pyspark.sql.types import IntegerType,BooleanType,DateType,StringType,DoubleType

# Convert String to Integer Type
df_spark.withColumn("age",df_spark.age.cast(IntegerType()))
df_spark.withColumn("age",df_spark.age.cast('int'))
df_spark.withColumn("age",df_spark.age.cast('integer'))
df_spark.withColumn("age",col("age").cast(StringType()))
df_spark.withColumn("age",col("age").cast(IntegerType()))
df_spark.withColumn("age",col("age").cast(DoubleType())).show(1)
#df_spark.withColumn("isGraduated",col("isGraduated").cast(BooleanType()))
#df_spark.withColumn("jobStartDate",col("jobStartDate").cast(DateType()))


# Using select
df_spark.select(col("age").cast('int').alias("age"))

#Using selectExpr()
df_spark.selectExpr("cast(age as int) age")

#Using with spark.sql()
#spark.sql("SELECT INT(age),BOOLEAN(isGraduated),DATE(jobStartDate) from CastExample")

#Date specific format

from pyspark.sql.functions import *

#df_pyspark.select(df_pyspark["Datecheck"], to_date(column("Datecheck"), "dd-MM-yyyy").
alias("Datecheckresult")).show()
```

```
+-----+----+----------+------+
| Name| age|Experience|Salary|
+-----+----+----------+------+
|Krish|31.0|        10| 30000|
+-----+----+----------+------+
only showing top 1 row
```

In [119]:

```
#pandas

df_pandas['age'].astype('int')
df_pandas['age'].astype('float')
df_pandas['age'].astype('str')[1]
#df_pandas["Date_of_joining"] = pd.to_datetime(df_pandas['Date_of_joining'])
#df_pandas["Jan Units"] = pd.to_numeric(df_pandas['Jan Units'], errors='coerce')
# coerce will replace all non-numeric values with NaN. ignore ignores
```

Out[119]:

```
'30'
```

In [120]:

```
#Get Dataframe rows that match a substring

#pyspark

df_spark.filter(df_spark.Name.contains("Sudh")).show(1)
df_spark.where(df_spark.Name.contains("Sudh")).show(1)

#pandas

df_pandas[df_pandas['Name'].str.contains('Sudh')]
```

```
+---------+---+----------+------+
|     Name|age|Experience|Salary|
+---------+---+----------+------+
|Sudhanshu| 30|         8| 25000|
+---------+---+----------+------+

+---------+---+----------+------+
|     Name|age|Experience|Salary|
+---------+---+----------+------+
|Sudhanshu| 30|         8| 25000|
+---------+---+----------+------+
```

Out[120]:

	Name	age	Experience	Salary	retirement age	eligible	extra bonus	Sex	Sex age
1	Sudhanshu	30	8	25000	33	Yes	28000	Male	Male 30

In [121]:

```
### sorting

#pyspark

df_spark.orderBy('age', ascending=False).show(2)

#pandas

df_pandas.sort_values(by='age', ascending=False)[0:2]
```

```
+---------+---+----------+------+
|     Name|age|Experience|Salary|
+---------+---+----------+------+
|    Krish| 31|        10| 30000|
|Sudhanshu| 30|         8| 25000|
+---------+---+----------+------+
only showing top 2 rows
```

Out[121]:

	Name	age	Experience	Salary	retirement age	eligible	extra bonus	Sex	Sex age
0	Krish	31	10	30000	34	Yes	33000	Male	Male 31
1	Sudhanshu	30	8	25000	33	Yes	28000	Male	Male 30

In [ ]:

```

```

In [ ]:

```

```